PhUSE US Connect 2018

Paper TT12

Breaking away from shell scripts for batch mode submissions on SAS® GRID

Robert Diseker, Chiltern- a Covance Company, Wilmington, NC, USA

ABSTRACT

Some SAS® Grid platforms restrict users from the command line and many a programmer has shirked at having to write a shell script to batch submit programs. A macro suite was developed that allows a user to submit SAS programs in batch mode and monitor their status on Grid using an interactive SAS Enterprise Guide session. This macro suite is used to create a "run-all" type SAS program that controls if programs are submitted sequentially or monitor multiple jobs submitted to maximize the processing power of the Grid. All of this can be performed from within an interactive session and the run-all programs can also be submitted as a scheduled job. If you are ready to break away from shell scripts and harness the power of Grid processing, please continue reading.

INTRODUCTION

When a statistical programming group recently moved to a SAS Grid on Linux with a SAS Enterprise Guide (EG) interface through an Ericom portal, there was a lot of concern how the large group who was accustomed to working in PC SAS in Windows would transition its workforce and processes with as little interruption as possible. The goal was, at a minimum, to figure out how to mirror what they were doing for batch submission of files in PC SAS all the while taking advantage of the load balancing and power of the SAS Grid. The group was accustomed to using windows based .bat files to submit sequential programs in batch mode but the SAS Grid did not support this. The use of shell scripts [1] was an option, but the group wanted to avoid the use of the command line interface. There were options to batch submit jobs to the grid and monitor them via the SAS Information Delivery Portal [2] or SAS Management Console, but the team chose to create a process that was more in line with the familiar PC interface, allowing programmers to submit programs for batch and monitor them within an interactive EG session.

A macro suite was developed that allows a user to submit SAS programs in batch mode and monitor their status on Grid using an interactive SAS EG session. The three main components for the process are defined as the ability to a) submit a single program in batch mode as a server job b) monitor the progress of the server job c) limit execution of SAS code until the server job completes. Included in the macro suite is the ability to create a "run-all" type program that controls whether programs are submitted sequentially or at once with monitoring to maximize the processing power of the Grid. All of this can be performed from within an interactive SAS Enterprise Guide session and the run-all programs can also be submitted as a scheduled job.

THE MACRO SUITE

The concept of the macro suite was to give the programmers comparable tools for production programming as they had in PC SAS. The macros were written to allow the programmer to have as much flexibility as possible within their programming project to ensure dependencies were met as well as optimize processing. The objectives for the macro suite were to:

a) submit a single program in batch mode
b) submit a group of programs sequentially in batch mode
c) control the order programs execute when submitted in batch mode
d) monitor the completion of a single job or multiple jobs on the SAS grid
e) create a "runall" of all programs in a directory for direct, sequential or customized processing

While the team had some help to get the first objective met, it later built onto this to address the other needs and round out the suite. What was achieved was a more powerful and customizable macro suite than had been anticipated.

There are three building blocks of the macro suite that had to be worked out to meet the objectives. The team worked through these in order as a) submit a single program in batch mode as a server job b) monitor the progress of the server job c) limit execution of SAS code until the server job completes.

SUBMITTING A SINGLE PROGRAM IN BATCH MODE AS A SERVER JOB

A contracted company provided the first macro called *%gsub*. This macro uses the path for the SAS version in a filename statement and takes advantage of a pipe command to call the SASGSUB command [3] on the Linux server. The SASGSUB batch submission via pipe command writes a .log and .lst file with the called program name to the same directory where the program is called. The results of the batch submission are displayed in the interactive EG log and a dataset is written to the work directory. This dataset, named GSUB, contains the job identifier (JOBID) that is unique to the program and time when it was submitted to the Grid server. The macro is simply called with the directory path and program name. It expects an *autoexec.sas* file to be located in the folder where the program resides (same directory path) and writes a .log and .lst file in the same folder. *Autoexec.sas* calls *setup.sas* that sets the options and *libnames* and other environmental macro variables for the program to execute correctly for a given project. Later, the macro was modified to use a single autoexec.sas in an upper directory to reduce the inherit risk of having multiple copies of *autoexec.sas* in a project area.

The macro call requires a full directory path and existing program name. Submit the program in a separate program from that called. Some users may want to store the *%gsub* call in the program itself, but it should not be called at the end of the program or the log will contain errors.

```
%gsub({directory path}/program.sas)
```

The major components of the source code include piping the SASGSUB command into a work.GSUB dataset. Set the macro variable SASGSUB to call the sasgsub batch submission utility for SAS version 9.4 on SAS GRID.

```
if appserver = "'SASApp94'" then call symput( 'SASGSUB',
    '/apps/sas/SAS94/Config_GCS/Lev1/Applications/SASGridManagerClientUtility/9.4/sa
    sgsub');
```

The pipe command for SASGSUB is as follows. Note that macro variable PGM is the full path and program name, PLOG and PLST are the paths to the program log and lst files that the SASGSUB command will create. The AEPATH macro variable is the path and name of the autoexec.sas file. The quotes must be exact, or the pipe command will not work.

```
filename gsub pipe "&SASGSUB -gridsubmitpgm '&pgm' -gridsasopts ""'-altlog '&plog'
              -altprint '&plst' -autoexec `&aepath' '"" ";
```

Once submitted, via the SASGSUB pipe command, it creates the GSUB dataset from the infile statement.

```
data gsub (drop= gsub_output len);
    infile gsub LRECL=500 MISSOVER dlm=',';
    length gsub_output $300 jobid len 8 jobdir $200 logfile $200;
    retain jobid jobdir logfile;
    input gsub_output ;
    if substr(gsub_output, 1, 6) = 'Job ID' then jobid = input( substr(gsub_output,
           17, 6), 6.);
    else if substr(gsub_output, 1, 13) = 'Job directory' then do;
           len = length( gsub_output ) - 17;
           jobdir = substr(gsub_output, 17,len);
    end;
    else if substr(gsub_output, 1, 12) = 'Job log file' then do;
           len = length( gsub_output ) - 17;
           logfile = substr(gsub_output, 17,len);
    output;
    end;
run;
```

The results are printed to the SAS EG log.

```
data _null_;
  set gsub;
  file print;
  put "Autoexec.sas used: &aepath";
  put "SAS Log: &plog";
  put "SAS Listing: &plst";
  put;
```

```
  put "Batch Job Submission Information";
  put;
  put 'Job ID:        ' jobid;
  put 'Job Directory: ' jobdir;
  put 'Job Log File:  ' logfile ;
run;
```

MONITOR THE PROGRESS OF THE SAS JOB

Jobs can be monitored from the SAS Grid servers using a pipe command BJOBS. The Platform Load Sharing Facility (LSF) has commands that are familiar to command line and IT users [4]. These allow for scheduling, execution and tracking of batch jobs. By using a pipe command, these can be called directly from EG and the output returned to a dataset for future processing. The BJOBS command pipes information of the user's server jobs into a local JOBS dataset.

```
filename jobs pipe "bjobs -a";
data jobs;
infile jobs firstobs=1 dlm=" " missover;
       length job_id $20. user $20. status $20. queue $20. sub_server $20.
ex_server
       $20. jobname $20. month $20. day $20. time $20.;
       input job_id $ user $ status $ queue $ sub_server $ ex_server $ jobname $
       month $ day $ time $;
       if job_id = "JOBID" then delete;
       if job_id = "Setting" then delete;
run;
```

SUBMIT A GROUP OF PROGRAMS SEQUENTIALLY IN BATCH MODE

The second requirement was to limit execution of SAS code until the server job completes so that jobs could be submitted sequentially. In statistical programming, when submitting a group of Study Data Tabulation Model (SDTM) or Analysis Data Model (ADAM) dataset programs, often the programmer wants the primary datasets like the demographics (DM) or subject-level analysis dataset (ADSL) to complete processing before running the other programs. When using PC SAS, a .bat file was generated that contained all programs in a text file. The programmer would order them in the .bat file and, when activated, each program would run sequentially by default, each program completing before the next one started.

Things work a bit differently on the SAS Grid as it is optimized to run many programs at once. If you run a program in EG containing a list of *%gsub* calls, the Grid is so optimized that all the programs will run on the servers at one time, as fast as possible. This could result in the last submitted program finishing before a longer running program that was submitted before it. To sequentially run programs on the SAS Grid server, there is the option to add -GRIDWAIT into the SASGSUB pipe command before the –GRIDSUMBITPGM command [3]. This option does not allow any further processing of code until the job completes on the grid. As the macro *%gsub* was created by a contractor and further knowledge about this option came later, the macro was built to address the need for a method to monitor progress of specific jobs on the SAS Grid.

This led to the development of the *%chkjob* macro, appendix A, which pipes the BJOBS command into a file with the results of the jobs submitted by the user. The macro subsets the server jobs to the JOBID from the work.GSUB dataset created from the most recent *%gsub* submission, and returns the status of the active JOBID from the SAS Grid server. It continues to monitor the status every five seconds and delays further execution of SAS code until the job completes, similar to how -gridwait option works.

```
data _null_;
       rc = sleep(5,1);
run;
```

Once the job completes on the server with a status of RUN, EXIT or DONE, it will allow SAS to execute the next row of code. The main difference between *%chkjob* and the use of -gridwait in the pipe command is that the programmer can follow the status of the jobs execution by watching progress displayed in the EG log. The EG log informs the user of the number of iterations of checking the server and elapsed time and when the job completes.

An example of using %chkjob for sequential batch mode submission would be batch submitting ADSL.sas and, once it finishes, batch submit the ADAE.sas program:

```
%gsub(/sasdata/sponsor/adam/adsl.sas)
%chkjob;
%gsub(/sasdata/sponsor/adam/adae.sas)
```

With *%chkjob*, the team successfully mirrored the .bat file operation in PC SAS to sequentially submit programs in batch mode. While this macro may seem like a duplication of effort given the -gridwait option can be used in the SASGSUB pipe command, this was the building block for an even more powerful couple of macros.

SUBMIT A GROUP OF PROGRAMS BATCH MODE TO RUN FAST AND MONITOR THEM UNTIL THEY COMPLETE

Once the team was able to submit a job, monitor its progress and delay SAS code execution of a single job, next targeted was the ability to take full advantage of the load balancing and processing power of the SAS Grid to handle multiple jobs at once. For example, often there is a group of programs that all can be submitted at once and allowed to run as fast as possible without regards to finish order, e.g., table programs. In some processes, there is a need to further execute more jobs once a set of programs complete. For example, a final efficacy dataset may depend on all datasets running before it or the lead may wish to run a final Quality Control (QC) program after all the programs complete to check the log files for errors and warnings and provide a summary of the directory as part of quality control. To do this on the SAS Grid, the predecessor jobs must be monitored to determine when they finish to allow the run of the log check on finalized logs. This could be done by calling *%bjobs* again and again until the programmer sees that all jobs complete, but by building this into a macro process, the whole production could be automated and free up the programmer for other tasks. The next plan capitalized on the similar concepts in *%chkjob* but had to create two macros to accomplish it: *%appendgsub* and *%chkalljobs*.

MACRO TO CREATE LIST OF JOBS: %APPENDGSUB

The first macro builds a list of submitted JOBIDs from a succession of calls to *%gsub* that are running on SAS Grid concurrently. This macro *%appendgsub*, appendix B, concatenates the JOBID from the work.GSUB dataset of a single batch submission to a dataset named work.APPENDGSUB.

MACRO TO CHECK FOR PENDING JOBS ON THE SERVER: %CHKALLJOBS

The second step in the process is to monitor the server for all pending jobs a user submits. The macro *%chkalljobs*, appendix C, compares the list of JOBIDs created in APPENDGSUB to the pending jobs on the grid to determine when all jobs complete before proceeding safely to execute the next line of code, e.g., run the final efficacy dataset or call the final QC macro on the directory of log files. This macro does the following things:

1. Checks that work.APPENDGSUB exists
2. It uses a pipe command of BJOBS macro to check the server for user jobs.
3. Checks that APPENDGSUB holds a JOBID that exists on the server and counts the pending jobs
4. While there are pending jobs, the macro will check for the status on the server of the JOBIDs contained in the APPENDGSUB list.
5. The macro will delay checking the server depending on the number of pending jobs as follows:

Number of Pending Jobs	Seconds delay before checking the server
>100	50
>50	20
>20	10
>0	5

6. When the number of pending jobs reaches zero then the macro ends
7. The macro deletes work.APPENDGSUB at the end of execution.
8. The EG log informs the user of the number of pending jobs, the elapsed time, and the wait time to the next server check.

CREATE A “RUNALL” PROGRAM

The last objective for the macro suite was to have a macro that creates a SAS program for submitting the production programs, a “runall” program, for normal flow or sequential batch submission of all programs in a directory. To prevent the programmer from having to type rows and rows of *%gsub* calls for all the programs in a directory with *%chkjob* or *%appendgsub* between each call, the *%brunall* macro, appendix D, dynamically builds a SAS program from a directory of files for either a normal flow or sequential submission (5). The runall type program is created in the called directory and does 3 things:

1. Batch submit each SAS program using *%gsub*
2. Depending on the type of runall used, adds macros *%chkjob, %appendgsub*, and *%chkalljobs* to track the submission of SAS programs on the Grid
3. Calls final QC macro on the directory of log files

The macro is called using the full directory path, with a positional macro variable set to “s” for sequential or “t” for a text file-based run order using a default file name of run_order.txt or indicate a different filename using the filename parameter.

The attributes of three different types of *%brunall* calls (Normal Flow, Sequential, and Textfile)

Task	Normal Flow	Sequential	Textfile
Macro call example	*%brunall(directory/tables)*	*%brunall(directory/adam,s)*	*%brunall(directory/adam,t)* Text file default =run_order.txt or *%brunall(directory/adam,t, filename = dataorder.txt)* Text file = bespoke file name
Batch Mode on Grid	Each program runs as fast as possible	Each file runs in sequential order	Customized order, generally sequential, followed by Normal Flow.
Monitoring	*%appendgsub* after each *%gsub* to collect a list of all the current JOBIDs you execute on the servers.	*%chkjob* after each *%gsub* ensures each program finishes before the next one starts	After each *%gsub* either *%appendgsub* (default) or *%chkjob* after each call if “,s” is in the text file
	%chkalljobs after last *%gsub*		*%chkalljobs* after last *%appendgsub* or when “,w” is indicated on a file
Post Batch Completion	QC program to review logs	QC program to review logs	QC program to review logs
Example of Resulting File	_tables_runall.sas	_adam_S_runall.sas	_adam_T_run_order_runall.sas or _adam_T_dataorder_runall.sas

When using a text file, the macro expects a text file named *run_order.txt* in the directory called with the program names listed in the expected order of execution. Indicate “,s” next to the program name for a sequential batch submission. Indicate a “,w” as in “wait”, next to the program name in run_order.txt to “wait” until preceding submitted jobs complete until the next program gets batch submitted (the macro adds a *%chkalljobs* after last *%appendgsub*).

An example text file for an ADaM batch submission named run_order.txt indicates that the first two files submit sequentially (indicated with “s”), each finishing before the next starting. The next four programs are safe to run as fast as possible so a *%appendgsub* is added to each one and a *%chkalljobs* added after adlb.sas (marked with “w”) to monitor and halt further SAS processing until the four files complete. Finally, once all predecessor datasets complete, the final dataset, zadtte,sas is free to execute. The run_order.txt file is on the left and the created _adam_T_run_order_runall.sas is on the right.

```
01adsl.sas,s
adae.sas
adcm.sas
adfa.sas
adlb.sas,w
zadtte.sas
```

```
%let path = sasdata/sandbox/adam;
*****************************;
*** Start Batch Calls ******;
*****************************;
%gsub.(&path./01adsl.sas);
%chkjob;
%gsub.(&path./adae.sas);
%appendgsub;
%gsub.(&path./adcm.sas);
%appendgsub;
%gsub.(&path./adfa.sas);
%appendgsub;
%gsub.(&path./adlb.sas);
%appendgsub;
%chkalljobs;
*****************************;
%gsub.(&path./zadtte.sas);
%appendgsub;
*******************************;
%chkalljobs;
*******************************;
****  Create Final QC File ****;
*******************************;
%qcchk(&path.);
```

The resulting SAS® program file is written to the called directory and can be executed all at once with an interactive session or scheduled using the SAS EG scheduler to run behind the scenes to execute an entire directory or directories of programs without programmer assistance or monitoring.

SCHEDULING A JOB

If you save a EG project with the *autoexec.sas* and the program generated by the *%brunall*, one can schedule a production run using the EG scheduler. The *autoexec.sas* file must point to a setup.sas file that loads the sasautos, where the macro suite is stored. Under File/Scheduler, change the radio button at "Run only when user is logged on" to "Run whether User is logged on or not. On the Triggers tab, click on the default trigger and then click Edit to the Settings to set up the run schedule. Turn off Synchronize across time zones. It is necessary to embed the program into the project by right-clicking on the program within EG, select Properties and click on the Embed button (it will be grayed out after selected). One you click OK, save the Project and log completely out of EG. The project will be saved and the program will run according to the schedule. There may be some differences due to local EG setup.

CONCLUSION

The macro suite for SAS Grid users on EG allows for a variety of production batch submissions and not only mirrors what was done in PC SAS with .bat files, it brings more flexibility to the programmer to harness the power of the SAS Grid load balancing and multiple servers. The runall programs allow the user to control what, when, and how programs are submitted to the SAS Grid and provides a production level operation that ensures the sequence of programs follows any dependencies necessary. The logs inform the user of the monitoring status and provides a transparent and user-friendly delivery model without command line processing or use of the SAS Information Delivery Portal.

Some limitations of this model are that the EG log can get quite large for long runs. Using bespoke text files, the programmer should take care in switching back and forth between sequential and normal execution of programs and review the program for dependencies and unintended results before submitting. Understanding how each macro works will allow nearly any combination of execution within a given program. Local IT resources will be best to understand how to schedule runall batch submissions for the SAS Grid. The code for these macros has been provided for the reader's convenience, but it may need to be edited for the local SAS Grid configuration.

Some improvements to the macro suite would be to develop a macro to kill unwanted or runaway jobs on the SAS servers. The amount of documentation on how the LSF Platform interacts with SAS Grid configuration seems limited, but it may benefit to explore LSF Platform commands, such as how to schedule programs on the server.

REFERENCES

1. http://support.sas.com/documentation/cdl/en/gridref/67371/HTML/default/viewer.htm#p0bjesvjde359nn1bfikmlzfk80b.htm
2. Adolfo Lopez, SAS Grid Job Submission and Monitoring from the SAS Information Delivery Portal, Valence Health, Chicago, IL http://support.sas.com/resources/papers/proceedings13/245-2013.pdf
3. http://support.sas.com/documentation/cdl/en/gridref/67371/HTML/default/viewer.htm#n0tiy9ucwq0wg5n1d03wn46pj4dk.htm
4. https://www.ibm.com/support/knowledgecenter/en/SSETD4_9.1.2/lsf_kc_cmd_ref.html
5. https://github.com/bmritz/SAS_MACROS/blob/master/get_filenames.sas

ACKNOWLEDGEMENTS

I would like to thank the D-Wise for their contributions and PDS Statistical Programming staff and management for their input and testing of these macros to help increase their efficiency and robustness.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Robert Diseker
Chiltern, a Covance Company
Work Phone: 910-338-4770
Email: Robert.diseker@chiltern.com

APPENDIX A - MACRO CHKJOB:

```
%macro chkjob;
%global chkjobfail;
%let chkjobfail = ; %**reset to missing;

%if %sysfunc(exist(gsub)) %then %do;
    %put [chkjob] work.gsub exists;
    proc sql noprint;
           select count(*) into :cntjob
           from gsub;
    quit;
    %if &cntjob = 0 %then %do; %**if jobid is missing;
           %let chkjobfail = FAIL;
           %put ERROR: [chkjob] No work.gsub jobid identified;
           %goto exitpgm;
    %end;
%end;
%else %do; %**if work.gsub does not exist;
    %let chkjobfail = FAIL;
    %put %sysfunc(sysmsg()) The file does not exist. ;
    %goto exitpgm;
%end;
%***get JobID;
Data _null_;
    set gsub;
    call symputx('jobid', put(jobid,8.));
run;
%let jobid=&jobid;

%if &jobid ne %then %do;
**check that GSUB holds a JOBID that exists in batch;
           %bjobs;
           %let jobmatch = N;
           data _null_;
                  set jobs (where=(strip(job_id) = "&jobid")) end = eof;
                  if eof and _n_ = 1 then  jobmatch = "Y";
                  call symput('jobmatch' , jobmatch);
           run;

    %if &jobmatch = N %then %do;%**Job does not exist;
           %let chkjobfail = FAIL;
           %put ERROR: [chkjob] work.GSUb jobid &jobid does not exist on server.
GSUB may have failed.;
           %symdel jobid;
           %goto exitpgm;
    %end;
    %let status = PEND;
    %let loop = 1;
    %let _start = %sysfunc(datetime());
    %do %until (%upcase(&status) = DONE | %upcase(&status) = EXIT);
           %bjobs;
           %let _end = %sysfunc(datetime());

           data _null_;
                  set jobs end = eof;
                  if  strip(job_id) = "&jobid" then do;
                         call Symputx('status' ,status);
                   if upcase(status) in ("DONE", "EXIT", "RUN") then
                      call symputx('jobname' ,jobname);
                   else call Symputx('jobname', ex_server);
                         put job_id status;
```

```
                end;
                if eof then do;
                       runtime = put((&_end - &_start),tod8.);
                       call symputx('_runtime',runtime);
                end;
          run;
          %put Job = &jobid.(%upcase(&jobname.)), loop #&loop., Time =
               %sysfunc(time(),tod8.), Elapsed time: &_runtime;
          %if &loop ne 1 %then %do;
                %***Wait 5 seconds before checking server again;
                data _null_;
                rc=SLEEP(5*1000);
                run;
          %end;
          %let loop = %eval(&loop +1);

    %end;
        %put ******************************************************************;
        %put [chkjob] Job Finished: &jobid.(%upcase(&jobname.)).  macro ended. ;
        %put ******************************************************************;
%end;   %*Job exists check;
%else %do;
    %let chkjobfail = FAIL;
    %symdel jobid;
    %put ERROR: [chkjob] JOBID is not available. GSUB did not submit. ;
%end;
%exitpgm: %**exit the program;
%put [chkjob] macro ended.;
%mend chkjob;
```

APPENDIX B - MACRO APPENDGSUB

```
%macro appendgsub;
%if %sysfunc(exist(gsub)) %then %do;
    %put [appendgsub] work.gsub exists;

    %if %sysfunc(exist(appendgsub)) %then %do;
    data appendgsub;
          set appendgsub gsub (in=a);
          if a then call symputx('jobid', jobid);
    run;
    proc sort data= appendgsub nodupkey;
          by jobid;
    run;
    %end;
    %else %do;
    data appendgsub;
          set gsub;
          call symputx('jobid', jobid);
    run;
    %end;
    %put Note: [appendgsub] Jobid &jobid was appended to APPENDGSUB;
%end;
%else %do; %**if work.gsub does not exist;
    %put Note: [appendgsub] %sysfunc(sysmsg()) The GSUB file does not exist. ;
%end;
%mend appendgsub;
```

APPENDIX C - MACRO CHKALLJOBS

```
%macro chkalljobs;
%global chkjobfail;
%let chkjobfail = ; %**reset to missing;

%if %sysfunc(exist(appendgsub)) = 0 %then %do;  %**if work.appendgsub does not
exist;
    %let chkjobfail = FAIL;
    %put %sysfunc(sysmsg()) The work.APPENDGSUB file does not exist. ;
    %goto exitpgm;
%end;
%else %do;
    %put [chkalljobs] work.appendgsub exists;
**check that GSUB holds a JOBID that exists in batch and how many =pendingjobs;
    %bjobs;
          proc sql noprint;select count(*) into :pendingjobs from
                (
                select b.jobid, a.status
                from jobs a inner join appendgsub b
                on input(a.job_id,8.) = b.jobid
                where upcase(a.status) not in ("DONE", "EXIT", "RUN")
                );
          quit;
          %put Note: [chkalljobs] Number of pending jobs = %cmpres(&pendingjobs);

    %if &pendingjobs  = 0 %then %do;%**No Job exists on server;
          %let chkjobfail = FAIL;
          %put Note: [chkalljobs] work.APPENDGSUB does not have any pending jobids
                     on server.;
          %***Delete the appendgsub dataset;
          proc datasets nolist;
                delete appendgsub;
          quit;
          %goto exitpgm;
    %end;
    %let loop = 1;
    %let _start = %sysfunc(datetime());
    %do %until (%cmpres(&pendingjobs) = 0);
          %bjobs;
          %let _end = %sysfunc(datetime());
          %**count of jobs on server that are pending;
          proc sql noprint;
                create table commonjobs as
                select b.jobid, a.status
                from jobs a inner join appendgsub b
                on input(a.job_id,8.) = b.jobid
                where upcase(a.status) not in ("DONE", "EXIT");

                select count(*) into :pendingjobs from commonjobs;
          quit;

          data _null_;
                runtime = put((&_end - &_start),tod8.);
                call symputx('_runtime',runtime);
          run;

          %***Wait xx seconds before checking server again;
          %if %cmpres(&pendingjobs) > 100 %then %let sec = 50;
          %else %if %cmpres(&pendingjobs) > 50 %then %let sec = 20;
          %else %if %cmpres(&pendingjobs) > 20 %then %let sec = 10;
          %else %let sec = 5;
          %if %cmpres(&pendingjobs.) > 0 %then %do;
```

```
            %put Note: [chkalljobs] loop #&loop., Time =
                 %sysfunc(time(),tod8.), Elapsed time: &_runtime.  Number of
                 pending jobs = %cmpres(&pendingjobs.);
            %put Note: [chkalljobs] Delay time to next server check = &sec.
                       seconds;
                data _null_;
                rc=SLEEP(1000*&sec.);
                run;
        %end;
        %let loop = %eval(&loop +1);
    %end;
        %***Delete the appendgsub dataset;
        proc datasets nolist;
            delete appendgsub commonjobs;
        quit;

      %put ************************************************************;
      %put Note: [chkalljob] All AppendGSUB JOBIDs Finished, AppendGSUB dataset
                 deleted , Elapsed time: &_runtime.;
      %put ************************************************************;
%end;   %*Job exists check;
%exitpgm: %**exit the program;
%put Note: [chkalljobs] macro ended.;
%mend chkalljobs;
```

APPENDIX D – MACRO BRUNALL

```
%macro brunall (path, inorder, filename=%str(run_order.txt);

%***what type of calls;
%let inorder = %upcase(&inorder);
%if &inorder = T and %sysfunc(fileexist(&path./&filename.)) %then
    %do;
        proc import datafile="&path./&filename." dbms=dlm out=work.runorder replace;
            getnames= no;
            guessingrows=500;
            delimiter=',';
        run;

        data filenames;
            set runorder;
            length filename $200 seq $20;
            filename = scan(var1,1,',');
            seq = scan(var2,1,',');
        run;
    %end;
%else %if &inorder = T %then %do;
    %put ERROR: filename does not exist. &path./&filename. ;
    %goto exit;
%end;
%else %do;
        %**normal and S runalls;
        %get_filenames(&path., out=filenames);

%end;  %**end of loop for normal and S runalls;

%let source = %scan (&path.,-1,/);

%local fname;
%if %upcase(&inorder.) = T %then %let fname = _%scan(&filename,1,".")_;

data yfiles2;
    length autocall $200;
```

```
    set filenames;
    if _n_ = 1 then do;
        autocall ="%"||"let path = &path.;";
        output;
        autocall ="***************************;";
        output;
        autocall ="*** Start Batch Calls ******;";
        output;
        autocall ="***************************;";
        output;
    end;

    if index(filename,".sas")>0 and index(filename, "autoexec")=0 and
         index(filename,"runall") = 0;

    autocall = "%"||"gsub.("||"&"||"path./"|| strip(filename)||");";
    output;

    **Sequential runall;
    %if %upcase(&inorder) = S %then %do;
        autocall = "%"||"chkjob;";
        output;
    %end;
    %else %if %upcase(&inorder) = T %then %do;
    **Textfile based runall;
        if upcase(compbl(seq)) = 'S' then do;
            autocall = "%"||"chkjob;";
            output;
        end;
        else if upcase(compbl(seq)) = 'W' then do;
            autocall = "%"||"appendgsub;";
            output;
            autocall = "%"||"chkalljobs;";
            output;
            autocall ="***************************;";
            output;
        end;
        else do;
            autocall = "%"||"appendgsub;";
            output;
        end;
    %end;
    %else %do;
    %***Basic Fast Runall;
            autocall = "%"||"appendgsub;";
            output;
    %end;
run;

data finappend;
    length autocall $200;
       autocall="*******************************;";
       output;
       autocall="%"||"chkalljobs;";
       output;
run;
data finalchk;
    length autocall $200;
       autocall="*******************************;";
       output;
       autocall="**** add qc checks ************;";
       output;
run;
```

```
%*** Check if text file is fully sequential;
%local numapp;
%if &inorder = T %then %do;
    proc sql noprint;
        select count(*) into :numapp
        from filenames
        where upcase(seq) ne 'S';
    quit;
%end;

data _null_;
    set  yfiles2 (keep= autocall)
        %if &inorder. ne S | (&inorder. = T & %cmpres(&numapp.) > 0) %then %do;
            finappend
        %end;
        finalchk
    ;
    file "&path./_&Source._&inorder.&fname.runall.sas" ;
    put autocall;
run;
%put NOTE: [brunall] Created Runall: "&path./_&Source._&inorder.&fname.runall.sas";
%exit:
%mend brunall;
```